If a group of subblocks must be compared so as to find all subsets of identical subblocks, the corresponding set of hashes of the subblocks may be calculated and compared instead.

Detailed Description Text (70):
Many of the uses of cryptographics hashes for subblocks can also be applied to blocks. For example, cryptographic hashes can be used to determine whether a block has changed at all since it was last backed up. Such a check could eliminate the need for further analysis.

Detailed Description Text (74):
This risk can be reduced or eliminated by calculating the hash (preferably a cryptographic hash of the block before it is partitioned into subblocks, storing the hash with an entity associated with the block as a whole, and then later comparing the stored hash with a compound hash of the reconstructed block. Such a check would provide a very strong safety net that would virtually eliminate the risk of undetected errors arising from the use of this invention.

Detailed Description Text (79):
We end this section with an example of a narrow hash function that has been implemented and tested and seems to perform well on a variety of data types. The hash function calculates a hash value from three bytes.

Detailed Description Text (98):
Multiple partitionings could also be useful simply to provide a wider pool of subblocks to compare. For example, it may be appropriate to partition each block of data in W different ways using W different functions F.sub.1 . . . F.sub.W where each function yields roughly the same subblock sizes, but at different positions within the block.

Detailed Description Text (100):
Comparing Subblocks

Detailed Description Text (102):
Compare the subblocks themselves.

Detailed Description Text (103):
Compare the hashes of the subblocks.

Detailed Description Text (104):
Compare identifies of the subblocks.

Detailed Description Text (105):
Compare references to the subblocks.

Detailed Description Text (114):
Of the multitude of solutions to the problem of matching blocks of data, one solution is worthy of special attention: the hash table. Hash tables consist of a (usually) finite array of slots into which values may be inserted. To add a value to a hash table, the value is hashed (using a hash function that is usually selected from the class of narrow hash functions) into a slot number, and the value is inserted into that slot. Later, the value can be retrieved in the same manner. Provisions must be made for the case where two data values, to be stored in the same table, hash to the same slot number.

Detailed Description Text (125):
Any component could carry additional checking information such as checksums or digests of the data in the component.

Detailed Description Text (138):
The processing of subblocks defined during a sequential partitioning of a block need not be deferred until the entire block has been partitioned. In particular, the hashes of already-defined subblocks could be calculated and compared while further subblocks are being defined.

Detailed Description Text (151):
Improving the efficiency of the hash calculations by using some part of the calculation of the has of the bytes at one position to calculate the hash at the next position. For example, it may be more efficient to calculate H(x,y,z) if H(*,x,y) has already been calculated. For

Freeform Search

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: random near (unique near identifier)

Display: 50 **Documents in Display Format:** FRO **Starting with Number** 1

Generate: ○ Hit List ◉ Hit Count ○ Side by Side ○ Image

Search Clear Interrupt

Search History

DATE: Wednesday, February 02, 2005 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=OR			
L20	L19 and (digital near sequence)	0	L20
L19	L18 and (unique near identifier)	4	L19
L18	(compar$ near random near number)	220	L18
L17	L16 and (digital near sequence$)	9	L17
L16	L15 and (unique near identifier)	645	L16
L15	Random near number	14182	L15
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L14	L3 and divid$	177	L14
L13	L3 and divid$	177	L13
L12	L4	177	L12
L11	L10 and (hallma$)	2	L11
L10	L6 and (phase near commit$)	43	L10
L9	L8 and (reserve near memory)	0	L9
L8	L6 and commit$	170	L8
L7	L6 and (free$ near memory)	7	L7
L6	L5 and (database near recover$)	266	L6
DB=USPT; PLUR=YES; OP=OR			
L5	707/$.ccls.	14279	L5

DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR

L4	L3 and divid$	177	L4
L3	L2 and uniform$	280	L3
L2	L1 and histogram	692	L2
L1	707/$.ccls.	24813	L1

END OF SEARCH HISTORY

rows of said third matrix storage means and at least a portion of an output from a prior modulo-2 addition, with an output from each said plurality of modulo-2 addition means providing as an output one of said row identifiers to one of said plurality of selection means and a portion of said ordered sequence of digital data until all of said ordered sequence of digital data is recovered.

Previous Doc Next Doc Go to Doc#

713/174, 180

L19: Entry 1 of 4 File: USPT Nov 30, 2004

DOCUMENT-IDENTIFIER: US 6826690 B1
TITLE: Using device certificates for automated authentication of communicating devices

Detailed Description Text (4):
FIG. 1B illustrates a representative interface device 32 with which a computer workstation 10 may communicate with other computer devices over a network. Such interface device, and the manner in which such devices operate, are well known in the art. A globally unique identifier of the interface device 32 (such as a medium access control, or "MAC", address) is stored in read-only memory 35 of the device 32. Data is sent and received over a communications link 38, which in the preferred embodiments is a LAN connection. Interface device 32 has also been augmented with additional features, as required by the present invention. As shown at 36, device 32 requires a protected storage or memory element. This protected storage 36 is used to securely store a private key 37 associated with dice 32. This protected storage 36 and private key 37 will be described in more detail below.

Detailed Description Text (21):
FIG. 4 depicts, at an abstract level, the relevant information used by the present invention for two representative paired devices. A client device 400 (which may actually function as a server in the distributed computing network, but which operates in the role of a client for tasks such as obtaining its IP address from a DHCP server) has a device certificate 410 stored therein. As discussed with reference to FIG. 3, this device certificate 410 includes a unique identifier 411 representing client device 400, where the value 431 of identifier 411 has been retrieved from a read-only memory 430 of a network interface adapter 32 attached to client device 400. The device certificate 410 also optionally contains capability flags 412, which for this client device are preferably set to the value "00" (as shown at 412a and 412b). A public key 413 is stored in the certificate 410 as well, and is cryptographically associated with (according to public key cryptography techniques) a private key value 421 which is stored in protected storage 420.

Detailed Description Text (22):
Client device 400 and server device 450 communicate, including exchange of their device certificates as appropriate, over communications link or network 38. Server device 450 has a device certificate 460, similar to certificate 410 of client device 400, where the value 481 of the device identifier 461 is the unique identifier which has been retrieved from read-only memory 480 of a network interface adapter 32 attached to server device 450. The server's device certificate 460 also optionally contains capability flags 462, which for this server device are shown as being set to the value "10" (as shown at 462a and 462b) to indicate that this device is an authorized address provider but is not an authorized DNS server. A public key 463 is stored in the certificate 460 as well, where this public key 463 is cryptographically associated with a private key value 471 which is stored in protected storage 470.

Detailed Description Text (54):
At Block 835, the client compares the random number 550 with the random number it previously created during Block 615 of FIG. 6. Block 840 asks whether the compared values were the same. If not, then this is not a trustworthy response, and it will be rejected by transferring control to Block 855. Otherwise, processing continues at Block 845.

CLAIMS:

32. The method as claimed in claim 31, wherein: said step of digitally signing said first message creates a first digital signature over one or more fields of said first message, wherein said one or more fields includes at least said globally-unique device identifier, and wherein said first device certificate is sent along with said digitally-signed first message;

said step of receiving said digitally-signed first message further comprises receiving said first device certificate, in addition to said digitally-signed first message; and said step of authenticating said first device further comprises the steps of: decrypting said first digital signature using said public key stored in said first device certificate; obtaining a certificate authority (CA) public key associated with a CA which created a second digital signature stored in said first device certificate; decrypting said second digital signature using said obtained CA public key; concluding that said first device certificate is authentic if said decrypted second digital signature is authentic; and concluding that said first device is authentic if (1) said decrypted first digital signature is authentic, (2) a device identifier value covered by said decrypted first digital signature matches a globally-unique identifier which identifies a sender of said first message, and (3) said first device certificate is authentic.

46. A computer-implemented method of using device certificates to authenticate communicating devices, comprising steps of: digitally signing a message, by a first device where said message is created, using a private key from a public key cryptography public/private key pair of the fist device thereby creating a digital signature for the message, wherein (1) a globally-unique identifier associated with a network adapter card of the first device is stored in a device certificate that is associated with the first device, thereby identifying the first device as an owner of the device certificate; (2) the public key is stored in the device certificate, thereby associating the public key with the globally-unique identifier; and (3) the private key is stored in device-resident, access-protected storage of the first device; and authenticating the first device, by a receiver of the digitally-signed message, before the receiver will act upon contents of the message, further comprising steps of: authenticating the it device as having created the digital signature on the digitally-signed message, by the receiver, using the public key of the first device; and ensuring that the globally-unique identifier stored in the device certificate matches a device identifier that identifies a sender of the digitally-signed message.

47. A computer-implemented method of using device certificates to authenticate communicating devices, comprising steps of: authenticating a first device, by a second device that receives a digitally-signed first message from the fist device, using a first device certificate of the first device; and responsive to a successful outcome of the authenticating step, responding to the first message, further comprising steps of: digitally signing a second message, by the second device where the second message is created, using a private key from a public key cryptography public/private key pair of the second device, thereby creating a digital signature for the second message, wherein (1) a globally-unique identifier associated with a network adapter card of the second device is stored in a second device certificate that is associated with the second device, thereby identifying the second device as an owner of the second device certificate; (2) the public key is stored in the second device certificate, thereby associating the public key with the globally-unique identifier; (3) the private key is stored in device-resident, access-protected storage of the second device; and (4) the second message includes the second device certificate, such that the digital signature on the second message covers the public key and the globally-unique identifier of the second device; authenticating the second device as having created the digital size on the digitally-signed second message, by the first device upon receipt thereof using the public key of the second device; and that the second message was sent to the first device by the second device by comparing the globally-unique identifier stored in the second device certificate to a device identifier that identifies a sender of the digitally-signed second message.

Previous Doc Next Doc Go to Doc#

L19: Entry 3 of 4 File: USPT Dec 30, 2003

DOCUMENT-IDENTIFIER: US 6671358 B1
TITLE: Method and system for rewarding use of a universal identifier, and/or conducting a financial transaction

Abstract Text (1):
A method for conducting a financial transaction corresponding to a good or service includes providing a plurality of first credit or debit devices; associating a universal identifier with each of the first credit or debit devices; associating one of the first credit or debit devices with a plurality of second credit or debit cards; employing the one of the first credit or debit devices to initiate the financial transaction; selecting one of the second credit or debit cards based upon the unique identifier and the good or service; and employing the selected one of the second credit or debit cards to conclude the financial transaction.

Brief Summary Text (10):
A Universal Identifier (UI) is a unique identifier for a particular person. Preferably, a UI is short enough to be memorable, yet long enough in order that all or most persons may have one or more UIs (e.g., for personal use, for business use) without duplication. One proposal for UIs is the use of a nickname (e.g., several words in length such as "Santa's Little Helper") along with an additional code called a checksum. Another proposal is a random combination of about eight letters and/or numbers, such as "A9B356JH".

Brief Summary Text (19):
These needs and others are met by the present invention, which employs a unique identifier, such as a universal identifier, to conduct a transaction, such as a financial transaction corresponding to a good or service.

Brief Summary Text (20):
As one aspect of the invention, a method for conducting a financial transaction corresponding to a good or service comprises: providing a plurality of first credit or debit devices; associating a unique identifier with each of the first credit or debit devices; associating one of the first credit or debit devices with a plurality of second credit or debit cards; employing the one of the first credit or debit devices to initiate the financial transaction; selecting one of the second credit or debit cards based upon the unique identifier and the good or service; and employing the selected one of the second credit or debit cards to conclude the financial transaction.

Brief Summary Text (21):
The method may include employing a universal identifier as the unique identifier. The method may further include providing a reward related to the universal identifier in response to the financial transaction.

Brief Summary Text (22):
As another aspect of the invention, a system for conducting a financial transaction comprises: means for storing a plurality of codes associated with goods or services; means for storing a plurality of unique identifiers each of which is associated with one of a plurality of first credit or debit devices; means for associating one of the first credit or debit devices with a plurality of second credit or debit cards; means for initiating the financial transaction for one of the goods or services with one of the first credit or debit devices; means for selecting one of the second credit or debit cards based upon the unique identifier of the one of the first credit or debit devices and the code of the one of the goods or services; and means for concluding the financial transaction with the selected one of the second credit or debit cards.

Detailed Description Text (11):
FIG. 2 shows a method for conducting a financial transaction corresponding to a good or service, such as 23. Each one of a plurality of unique identifiers, such as universal identifiers 12, is associated with a corresponding one of a plurality of first credit and/or debit devices, such as the exemplary credit or debit cards 14. In the exemplary embodiment, the exemplary first cards 14 selectively function as a credit card or a debit card, although the invention is applicable to first cards which function as only one of a credit card or a debit card. Each of the first cards 14, such as card 16, is associated with a plurality of second credit and/or debit devices, such as the exemplary credit or debit cards 18. In the exemplary embodiment, the exemplary second cards 18 function as only one of a credit card or a debit card, although the invention is applicable to such cards which function as both a credit card and a debit card. At 20, one of the first cards 14, such as card 16, is employed to initiate a financial transaction 22 corresponding to a good or service 23 of the goods 24 and services 26. Next, at 28, one of the second cards 18, such as card 29, is selected based upon the universal identifier (UID) 30 of the first card 16 and the good or service 23 of interest. Finally, at 31, the selected second card 29 is employed to conclude the financial transaction 22.

Detailed Description Text (70):
Portion 96 represents the flowchart of the gaming servers 49 of FIG. 3. At 126, the UID 102 is received by a suitable game engine. Next, at 128, a random number is selected by a suitable random number generator. Then, at 130, it is determined whether the random number is a winning number (e.g., by comparing the random number to a predetermined list of winning numbers). If so, then, at 132, the UID member 100 is informed (e.g., through an e-mail message) of the win. Otherwise, if there was no win, then, at 134, the UID member 100 is similarly informed of a loss.

CLAIMS:

1. A method for conducting a financial transaction corresponding to a good or service, said method comprising the steps of: providing a plurality of first credit or debit devices; associating a unique identifier with each of said first credit or debit devices; associating one of said first credit or debit devices with a plurality of second credit or debit cards; employing said one of said first credit or debit devices to initiate said financial transaction; selecting one of said second credit or debit cards based upon said unique identifier and said good or service; and employing said selected one of said second credit or debit cards to conclude said financial transaction.

2. The method of claim 1 further comprising: employing a universal identifier as said unique identifier.

7. The method of claim 1 further comprising: entering the unique identifier of said one of said first credit or debit devices on a web page.

8. The method of claim 1 further comprising: entering the unique identifier of said one of said first credit or debit devices on a wireless communication device.

15. A system for conducting a financial transaction, said system comprising: means for storing a plurality of codes associated with goods or services; means for storing a plurality of unique identifiers each of which is associated with one of a plurality of first credit or debit devices; means for associating one of said first credit or debit devices with a plurality of second credit or debit cards; means for initiating said financial transaction for one of said goods or services with one of said first credit or debit devices; means for selecting one of said second credit or debit cards based upon the unique identifier of said one of said first credit or debit devices and the code of said one of said goods or services; and means for concluding said financial transaction with said selected one of said second credit or debit cards.

Previous Doc Next Doc Go to Doc#

3. Document ID: US 20020194209 A1

L8: Entry 3 of 6 File: PGPB Dec 19, 2002

PGPUB-DOCUMENT-NUMBER: 20020194209
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020194209 A1

TITLE: On-disk file format for a serverless distributed file system

PUBLICATION-DATE: December 19, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Bolosky, William J.	Issaquah	WA	US	
Cermak, Gerald	Bothell	WA	US	
Adya, Atul	Bellevue	WA	US	
Douceur, John R.	Bellevue	WA	US	

US-CL-CURRENT: 707/205; 707/200

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw. Desc | Image

4. Document ID: US 20010037323 A1

L8: Entry 4 of 6 File: PGPB Nov 1, 2001

PGPUB-DOCUMENT-NUMBER: 20010037323
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20010037323 A1

TITLE: Hash file system and method for use in a commonality factoring system

PUBLICATION-DATE: November 1, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Moulton, Gregory Hagan	Irvine	CA	US	
Whitehill, Stephen B.	Tustin	CA	US	

US-CL-CURRENT: 707/1; 707/10, 711/216

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw. Desc | Image

5. Document ID: US 6704730 B2

L8: Entry 5 of 6 File: USPT Mar 9, 2004

US-PAT-NO: 6704730
DOCUMENT-IDENTIFIER: US 6704730 B2
**** See image for Certificate of Correction ****

TITLE: Hash file system and method for use in a commonality factoring system

DATE-ISSUED: March 9, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Moulton; Gregory Hagan	Irvine	CA		
Whitehill; Stephen B.	Tustin	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Avamar Technologies, Inc.	Irvine	CA			02

APPL-NO: 09/ 777150 [PALM]
DATE FILED: February 5, 2001

PARENT-CASE:
CROSS REFERENCE TO RELATED PATENT APPLICATIONS The present invention claims priority from United States Provisional Patent Application Serial No. 60/183,762 for: "System and Method for Decentralized Data Storage" filed Feb. 18, 2000, the disclosure of which is herein specifically incorporated by this reference.

INT-CL: [07] G06 F 17/30

US-CL-ISSUED: 707/6; 707/7, 707/204
US-CL-CURRENT: 707/6; 707/204, 707/7

FIELD-OF-SEARCH: 707/7, 707/6, 707/204, 714/798, 717/139, 341/87, 341/51

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
3668647	June 1972	Evangelisti et al.	340/172.5
4215402	July 1980	Mitchell et al.	364/200
4404676	September 1983	DeBenedictis	714/798
4649479	March 1987	Advani et al.	364/300
4761785	August 1988	Clark et al.	371/51
4887204	December 1989	Johnson et al.	364/200
4887235	December 1989	Holloway et al.	364/900
4897781	January 1990	Chang et al.	364/200
4901223	February 1990	Rhyne	364/200
4929946	May 1990	O'Brien et al	341/87
4982324	January 1991	McConaughy et al.	364/200
5005122	April 1991	Griffin et al.	364/200
5016009	May 1991	Whiting et al.	341/67
5018060	May 1991	Gelb et al.	364/200
5089958	February 1992	Horton et al.	395/575
5109515	April 1992	Laggis et al.	395/725
5126739	June 1992	Whiting et al.	341/106
5133065	July 1992	Cheffetz et al.	395/575
5140321	August 1992	Jung	341/55
5146568	September 1992	Flaherty et al.	395/325
5155835	October 1992	Belsan	395/425

5162986	November 1992	Graber et al.	364/146
51[illegible]48	November 1992	Walls	395/600
5210866	May 1993	Milligan et al.	395/575
5218695	June 1993	Noveck et al.	395/600
5239637	August 1993	Davis et al.	395/425
5239647	August 1993	Anglin et al.	395/600
5239659	August 1993	Rudeseal et al.	395/800
5263154	November 1993	Eastridge et al.	395/575
5276860	January 1994	Fortier et al.	395/575
5276867	January 1994	Kenley et al.	395/600
5278838	January 1994	Ng et al.	371/10.1
5281967	January 1994	Jung	341/55
5305389	April 1994	Palmer	382/1
5317728	May 1994	Tevis et al.	395/600
5325505	June 1994	Hoffecker et al.	395/425
5347653	September 1994	Flynn et al.	395/600
5355453	October 1994	Row et al.	395/200
5367637	November 1994	Wei	395/250
5367698	November 1994	Webber et al.	395/800
5379418	January 1995	Shimazaki et al.	395/575
5403639	April 1995	Belsan et al.	395/600
5404508	April 1995	Konrad et al.	395/600
5404527	April 1995	Irwin et al.	395/700
5406279	April 1995	Anderson et al.	341/51
5448718	September 1995	Cohn et al.	395/404
5452440	September 1995	Salsburg	395/463
5452454	September 1995	Basu	395/700
5454099	September 1995	Myers et al.	395/575
5479654	December 1995	Squibb	395/600
5487160	January 1996	Bemis	395/441
5497483	March 1996	Beardsley et al.	395/575
5513314	April 1996	Kandasamy et al.	395/182.04
5515502	May 1996	Wood	395/182.13
5521597	May 1996	Dimitri	341/51
5524205	June 1996	Lomet et al.	395/182.14
5535407	July 1996	Yanagawa et al.	395/800
5544320	August 1996	Konrad	395/200.09
5559991	September 1996	Kanfi	395/489
5574906	November 1996	Morris	395/601
5586322	December 1996	Beck et al.	395/616
5604862	February 1997	Midgely et al.	395/182.04
5606719	February 1997	Nichols et al.	395/876
5640561	June 1997	Satoh et al.	395/618
5649196	July 1997	Woodhill et al.	395/620
5659743	August 1997	Adams et al.	395/621
5659747	August 1997	Nakajima	395/651
5696901	December 1997	Konrad	395/200.09
5742811	April 1998	Agrawal et al.	707/6
5751936	May 1998	Larson et al.	395/182.05
5754844	May 1998	Fuller	707/6

5765173	June 1998	Cane et al.	707/204
5771354	June 1998	Crawford	395/200.59
5778395	July 1998	Whiting et al.	707/204
5794254	August 1998	McClain	707/204
5802264	September 1998	Chen et al.	395/182.04
5802297	September 1998	Engquist	395/200.42
5831558	November 1998	Harvell	341/106
5850565	December 1998	Wightman	315/821
5915024	June 1999	Kitaori et al.	713/176
5933104	August 1999	Kimura	341/87
5978791	November 1999	Farber et al.	707/2
5990810	November 1999	Williams	341/51
6014676	January 2000	McClain	707/204
6016553	January 2000	Schneider et al.	714/21
6029168	February 2000	Frey	707/10
6044220	March 2000	Breternitz, Jr.	717/139
6085298	July 2000	Ohran	711/162
6122754	September 2000	Litwin et al.	714/4
6141421	October 2000	Takaragi et al.	380/30
6170744	January 2001	Lee et al.	235/380
6219423	April 2001	Davis	380/268
6268809	July 2001	Saito	341/51
6307487	October 2001	Luby	
6320520	November 2001	Luby	
6377907	April 2002	Waclawski	702/182
6535894	March 2003	Schmidt et al.	707/204
6553494	April 2003	Glass	713/186

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
WO-96/25801	August 1996	WO	

OTHER PUBLICATIONS

Hegazy, A.E.F.A. "Searching Large Textual Files for Near Matching Patterns", Dissertation, School of Engineering and Applied Science, George Washington University, Jul. 24, 1985.*
Hegazy, A.E.F.A. "Searching Large Textual Files for Near Matching Patterns", Dissertation, School of Engineering and Applied Sciences, George Washington University, Jul. 24, 1985.

ART-UNIT: 2172

PRIMARY-EXAMINER: Corrielus; Jean M.

ATTY-AGENT-FIRM: Kubida; William J. Bernard; Eugene J. Hogan & Hartson LLP

ABSTRACT:

A system and method for a computer file system that is based and organized upon hashes and/or strings of digits of certain, different, or changing lengths and which is capable of eliminating or screening redundant copies of aggregate blocks of data (or parts of data blocks) from the system. The hash file system of the present invention utilizes hash values for computer files or file pieces which may be produced by a checksum generating program, engine or

algorithm such as industry standard MD4, MD5, SHA or SHA-1 algorithms. Alternatively, the hash values may be generated by a checksum program, engine, algorithm or other means that produces an effectively unique hash value for a block of data of indeterminate size based upon a non-linear probablistic mathematical algorithm.

45 Claims, 13 Drawing figures

Full | Title | Citation | Front | Review | Classification | Date | Reference | | | Claims | KWIC | Draw Desc | Image

---

6. Document ID: US 5990810 A

L8: Entry 6 of 6 File: USPT Nov 23, 1999

US-PAT-NO: 5990810
DOCUMENT-IDENTIFIER: US 5990810 A

TITLE: Method for partitioning a block of data into subblocks and for storing and communcating such subblocks

DATE-ISSUED: November 23, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Williams; Ross Neil	Adelaide SA5000			AU

APPL-NO: 08/ 894091 [PALM]
DATE FILED: August 15, 1997

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
AU	PN1232	February 17, 1995
AU	PN2392	April 12, 1995

PCT-DATA:

APPL-NO	DATE-FILED	PUB-NO	PUB-DATE	371-DATE	102(E)-DATE
PCT/AU96/00081	February 15, 1996	WO96/25801	Aug 22, 1996	Aug 15, 1997	Aug 15, 1997

INT-CL: [06] H03 M 7/00

US-CL-ISSUED: 341/51; 341/67
US-CL-CURRENT: 341/51; 341/67

FIELD-OF-SEARCH: 341/51, 341/50, 341/67, 375/241, 704/203

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
4698628	October 1987	Herkert et al.	340/825.02
5235623	August 1993	Sugiyama et al.	341/67
5479654	December 1995	Squibb	395/600

OTHER PUBLICATIONS

Williams, Ross, "An algorithm for matching text (possibly original)", Newsgroup posting, comp.compression, Jan. 27, 1992.
Williams, Ross, "Parallel data compression", Newsgroup posting, comp.conmpression.research, Jun. 30, 1992.
Knuth, Donald E., "The Art of Computer Programming, vol. 3: Sorting and Searching", pp. 508-513, Addison-Wesley Publishing Company, 1973.
Williams, Ross N., "An Introduction to Digest Algorithms" Proceedings of the Digital Equipment Computer Users Society, pp. 9-18, Aug. 1994.
Williams, Ross N., "An Extremely Fast ZIV-Lempel Data Compression Algorithm", Proceedings of Data Compression Conference, pp. 362-371, Apr. 1991.
Knuth, Donald E., The Art of Computer Programming, vol. 1: Fundamental Algorithms, pp. 435-451, Addison Wesley Publishing Company, 1973.

ART-UNIT: 289

PRIMARY-EXAMINER: Young; Brian

ATTY-AGENT-FIRM: Traurig; Greenberg Bell; Robert P.

ABSTRACT:

This invention provides a method and apparatus for detecting common spans within one or more data blocks by partitioning the blocks (FIG. 4) into subblocks and searching the group of subblocks (FIG. 12) (or their corresponding hashes (FIG. 13)) for duplicates. Blocks can be partitioned into subblocks using a variety of methods, including methods that place subblock boundaries at fixed positions (FIG. 3), methods that place subblock boundaries at data-dependent positions (FIG. 3), and methods that yield multiple overlapping subblocks (FIG. 6). By comparing the hashes of subblocks, common spans of one or more blocks can be identified without ever having to compare the blocks or subblocks themselves (FIG. 13). This leads to several applications including an incremental backup system that backs up changes rather than changed files (FIG. 25), a utility that determines the similarities and differences between two files (FIG. 13), a file system that stores each unique subblock at most once (FIG. 26), and a communications system that eliminates the need to transmit subblocks already possessed by the receiver (FIG. 19).

30 Claims, 26 Drawing figures

Full | Title | Citation | Front | Review | Classification | Date | Reference | | | Claims | KWIC | Draw Desc | Image

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Term	Documents
LIST	356097
LISTS	105838
HASH	19927
HASHES	2109
(7 AND (HASH NEAR LIST)).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	6
(L7 AND (LIST NEAR HASH)).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	6

- **Display Format**: - Change Format
- Previous Page Next Page Go to Doc#

Refine Search

Search Results -

Term	Documents
LIST	356097
LISTS	105838
HASH	19927
HASHES	2109
(7 AND (HASH NEAR LIST)).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	6
(L7 AND (LIST NEAR HASH)).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	6

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L8

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Wednesday, February 02, 2005 Printable Copy Create Case

Set Name side by side	**Query**	**Hit Count**	**Set Name** result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L8	L7 and (list near hash)	6	L8
L7	L6 and piece$	24	L7
L6	L4 and compar$	75	L6
L5	L3 and chunck$	1	L5
L4	L3 and (check$)	80	L4
L3	L2 and (hash$ near value)	90	L3
L2	L1 and (hash$ near file)	132	L2
L1	hash$ near algorithm$	3392	L1

END OF SEARCH HISTORY

both X and Y. For example, if E1 possessed the hashes of the subblocks of Y, it could compare them to the hashes of the subblocks of X to determine the subblocks common to both X and Y. Subblocks that are not common can be transmitted explicitly. Subblocks that are common to both X and Y can be transmitted by transmitting a reference to the subblock.

Brief Summary Text (115):
26. In a further aspect of the invention, the invention provides a method according to one of aspects 1 to 6, wherein said subblocks are compared by comparing the hashes of said subblocks.

Drawing Description Text (13):
FIG. 12 depicts a method (and apparatus) for partitioning two blocks b1 and b2 into subblocks, using a constraint F, and then comparing the subblocks.

Detailed Description Text (16):
The identity of a subblock means any piece of information that could be used in place of the subblock for the purpose of comparison for identicality. Identities include, but are not limited to:

Detailed Description Text (19):
The subblock acts as its own identity because subblocks themselves can be compared with each other. Hashes of subblocks also act as identities of subblocks because hashes of subblocks can be compared with each other to determine if their corresponding subblocks are identical.

Detailed Description Text (20):
A reference to a subblock means any piece of information that could be used in practice by one entity to identify to another entity (or itself) a particularly valued subblock, where the two entities may already share some knowledge. For example, the two entities might each possess the knowledge that the other entity already possesses ten subblocks of known values having particular index values numbered one to ten.

Detailed Description Text (47):
Narrow hash functions: Narrow hash functions are the weakest class of hash functions and generate output values that are so narrow (e.g. 16 bits) that the entire space of output values can be searched in a reasonable amount of time. For example, an 8-bit hash function would map any data block to a hash in the range 0 to 155. A 16-bit has function would map to a hash in the range 0 to 65535. Given a particular hash value, it would be possible to find a corresponding block simply by generating random blocks and feeding them into the narrow hash function until the searched-for value appeared. Narrow hash functions are usually used to arbitrarily (but deterministically) classify a set of data values into a small number of groups. As such, they are useful for constructing hash table data structures, and for detecting errors in data transmitted over noisy communication channels. Examples of this class: CRC-16, CRC-32. Fletcher checksum, the IP checksum.

Detailed Description Text (48):
Wide hash functions: Wide hash functions are similar to narrow hash functions except that their output values are significantly wider. At a certain point this quantitative difference implies a qualitative difference. In a wide hash function, the output value is so wide (e.g. 128 bits) that the probability of any two randomly chosen blocks having the same hashed value is negligible (e.g. about one in 10.sup.38). This property enables these wide hashes to be used as "identities" of the blocks of data from which they are calculated. For example, if entity E1 has a block of data and sends the wide hash of the block to an entity E2, then if entity E2 has a block that has the same hash, then the a-priori probability of the blocks actually being different is negligible. The only catch is that wide hash functions are not designed to be non-invertible. Thus, while the space of (say)2.sup.128 values is too large to search in the manner described for narrow hash functions, it may be easy to analyse the hash function and calculate a block corresponding to a particular hash. Accordingly, E1 could fool E2 into thinking E1 had one block when it really had a different block. Examples of this class: any 128-bit CRC algorithm.

Detailed Description Text (49):
Weak one-way hash functions: Weak one-way hash functions are not only wide enough to provide "identity", but they also provide cryptographic assurance that it will be extremely difficult, given a particular hash value, to find a block corresponding to that hash value. Examples of

L8: Entry 6 of 6 File: USPT Nov 23, 1999

DOCUMENT-IDENTIFIER: US 5990810 A

TITLE: Method for partitioning a block of data into subblocks and for storing and communcating such subblocks

Abstract Text (1):
This invention provides a method and apparatus for detecting common spans within one or more data blocks by partitioning the blocks (FIG. 4) into subblocks and searching the group of subblocks (FIG. 12) (or their corresponding hashes (FIG. 13)) for duplicates. Blocks can be partitioned into subblocks using a variety of methods, including methods that place subblock boundaries at fixed positions (FIG. 3), methods that place subblock boundaries at data-dependent positions (FIG. 3), and methods that yield multiple overlapping subblocks (FIG. 6). By comparing the hashes of subblocks, common spans of one or more blocks can be identified without ever having to compare the blocks or subblocks themselves (FIG. 13). This leads to several applications including an incremental backup system that backs up changes rather than changed files (FIG. 25), a utility that determines the similarities and differences between two files (FIG. 13), a file system that stores each unique subblock at most once (FIG. 26), and a communications system that eliminates the need to transmit subblocks already possessed by the receiver (FIG. 19).

Brief Summary Text (8):
To identify identical portions of data within a group of blocks of data, the blocks must be analysed. One simple approach is to divide the blocks into fixed-length (e.g. 512-byte) subblocks and compare these with each other so as to identify all identical subblocks. This knowledge can the be used to manage the blocks in more efficient ways.

Brief Summary Text (19):
Low-redundancy file system: Data stored in a file system can be partitioned into subblocks whose hashes can be compared so as to eliminate the redundant storage of identical subblocks.

Brief Summary Text (22):
The term block and subblock both refer, without limitation, to finite blocks or infinite blocks (sometimes called streams) of zero or more bits or bytes of digital data. Although the two different terms ("blocks" and "subblock") essentially describe the same substance (digital data), the two different terms have been employed in this specification to indicate the role that a particular piece of data is playing. The term "block" is usually used to refer to raw data to be manipulated by aspects of the invention. The term "subblock" is usually used to refer to a part of a block. "Blocks" are "partitioned" into "subblocks".

Brief Summary Text (44):
Note: In most applications, the output of this aspect will be an ordered list of hashes of the subblocks of the block.

Brief Summary Text (45):
9. In a further aspect of the invention, the invention provides a method for comparing one or more blocks, comprising the steps of:

Brief Summary Text (48):
c. Comparing the elements of said projections of said blocks.

Brief Summary Text (64):
Note: It is implicit in this aspect the E1 will be able to use comparison (or other methods) to use its knowledge of E2's possession of Y to determine the set of subblocks that are common to

both X and Y. For example, if E1 possessed the hashes of the subblocks of Y, it could compare them to the hashes of the subblocks of X to determine the subblocks common to both X and Y. Subblocks that are not common can be transmitted explicitly. Subblocks that are common to both X and Y can be transmitted by transmitting a reference to the subblock.

Brief Summary Text (115):
26. In a further aspect of the invention, the invention provides a method according to one of aspects 1 to 6, wherein said subblocks are compared by comparing the hashes of said subblocks.

Drawing Description Text (13):
FIG. 12 depicts a method (and apparatus) for partitioning two blocks b1 and b2 into subblocks, using a constraint F, and then comparing the subblocks.

Detailed Description Text (16):
The identity of a subblock means any piece of information that could be used in place of the subblock for the purpose of comparison for identicality. Identities include, but are not limited to:

Detailed Description Text (19):
The subblock acts as its own identity because subblocks themselves can be compared with each other. Hashes of subblocks also act as identities of subblocks because hashes of subblocks can be compared with each other to determine if their corresponding subblocks are identical.

Detailed Description Text (20):
A reference to a subblock means any piece of information that could be used in practice by one entity to identify to another entity (or itself) a particularly valued subblock, where the two entities may already share some knowledge. For example, the two entities might each possess the knowledge that the other entity already possesses ten subblocks of known values having particular index values numbered one to ten.

Detailed Description Text (47):
Narrow hash functions: Narrow hash functions are the weakest class of hash functions and generate output values that are so narrow (e.g. 16 bits) that the entire space of output values can be searched in a reasonable amount of time. For example, an 8-bit hash function would map any data block to a hash in the range 0 to 155. A 16-bit has function would map to a hash in the range 0 to 65535. Given a particular hash value, it would be possible to find a corresponding block simply by generating random blocks and feeding them into the narrow hash function until the searched-for value appeared. Narrow hash functions are usually used to arbitrarily (but deterministically) classify a set of data values into a small number of groups. As such, they are useful for constructing hash table data structures, and for detecting errors in data transmitted over noisy communication channels. Examples of this class: CRC-16, CRC-32. Fletcher checksum, the IP checksum.

Detailed Description Text (48):
Wide hash functions: Wide hash functions are similar to narrow hash functions except that their output values are significantly wider. At a certain point this quantitative difference implies a qualitative difference. In a wide hash function, the output value is so wide (e.g. 128 bits) that the probability of any two randomly chosen blocks having the same hashed value is negligible (e.g. about one in 10.sup.38). This property enables these wide hashes to be used as "identities" of the blocks of data from which they are calculated. For example, if entity E1 has a block of data and sends the wide hash of the block to an entity E2, then if entity E2 has a block that has the same hash, then the a-priori probability of the blocks actually being different is negligible. The only catch is that wide hash functions are not designed to be non-invertible. Thus, while the space of (say)2.sup.128 values is too large to search in the manner described for narrow hash functions, it may be easy to analyse the hash function and calculate a block corresponding to a particular hash. Accordingly, E1 could fool E2 into thinking E1 had one block when it really had a different block. Examples of this class: any 128-bit CRC algorithm.

Detailed Description Text (49):
Weak one-way hash functions: Weak one-way hash functions are not only wide enough to provide "identity", but they also provide cryptographic assurance that it will be extremely difficult, given a particular hash value, to find a block corresponding to that hash value. Examples of

this class: a 64-bit DES hash.

Detailed Description Text (50):
Strong one-way has functions: Strong one-way hash functions are the same as weak one-way hash functions except that they have the additional property of providing cryptographic assurance that it is difficult to find any two different blocks that have the same hash value, where the hash value is unspecified. Examples of this class: MD4, MD5, and SHA-1.

Detailed Description Text (56):
Depending on the application, hash functions from any of the four classes above could be employed in either role. However, as the determination of subblock boundaries does not require identity or cryptographic strength, it would be inefficient to use hash functions from any but the weakest class. Similarly, the need for identity, the ever-present threat of subversion, and the minor performance penalty for strong one-way hash functions (compared to weak ones) suggests that nothing less than strong one-way hash functions should be used to calculate subblock identities.

Detailed Description Text (61):
The theoretical properties of cryptographic hashes (and here is meant strong one-way hash functions) yield particularly interesting practical properties. Because such hashes are significantly wide, the probability of two randomly-chosen subblocks having the same hash is practically zero (for a 128-bit hash, it is about one in 10.sup.38), and because it is computationally infeasible to find two subblocks having the same hash, it is practically guaranteed that no intelligent agent will be able to do so. The implication of these properties is that from a practical perspective, the finite set of hash values for a particular cryptographic hash algorithm is one-to-one with the infinite set of finite variable length subblocks. This theoretically impossible property manifests itself in practice because of the practical infeasibility of finding two subblocks that hash to the same value.

Detailed Description Text (62):
This property means that, for the purposes of comparison (for identically), cryptographic hashes may safely be used in place of the subblocks from which they were calculated. As most cryptographic hashes are only about 128 bits long, hashes provide an extremely efficient way to compare subblocks without requiring the direct comparison of the content of the subblocks themselves. Such comparisons can be used to eliminate many transmissions of information. For example, a subblock X.sub.1 on a computer C1 in Sydney could be compared with a subblock Y.sub.1 on a computer C2 in Boston by a computer C3 in Paris, with the total theoretical network traffic being just 256 bits (C1 and C2 each send the 128-bit hash of their respective subblocks to C3 for comparison, and C3 compares the two hashes).

Detailed Description Text (64):
Cryptographic hashes can be used to compare two subblocks without having to compare, or requiring access to, the content of the subblocks.

Detailed Description Text (65):
If it is necessary to be able to determine whether a subblock T is identical to one of a group of subblocks, the subblocks themselves need not be stored, just a collection of their hashes. The hash of any candidate subblock can then be compared with the hashes in the collection to establish whether the subblock is in the group of subblocks from which the collection of hashes was generated.

Detailed Description Text (66):
Cryptographic hashes can be used to ensure that the partitioning of a block into subblocks and the subsequent reassembly of the subblocks into a reconstructed block is error-free. This can be done by comparing the hash of the original block with the hash of the reconstructed block.

Detailed Description Text (68):
If an entity E1 passes a key (consisting of a block of bits) chosen at random to an entity E2, E2 may then prove to E1 that it possesses a subblock by sending E1 the hash of the concatenation of the key and the subblock. This mechanism could be used as an additional check in security applications.

Detailed Description Text (69):

If a group of subblocks must be compared so as to find all subsets of identical subblocks, the corresponding set of hashes of the subblocks may be calculated and compared instead.

Detailed Description Text (70):
Many of the uses of cryptographics hashes for subblocks can also be applied to blocks. For example, cryptographic hashes can be used to determine whether a block has changed at all since it was last backed up. Such a check could eliminate the need for further analysis.

Detailed Description Text (74):
This risk can be reduced or eliminated by calculating the hash (preferably a cryptographic hash of the block before it is partitioned into subblocks, storing the hash with an entity associated with the block as a whole, and then later comparing the stored hash with a compound hash of the reconstructed block. Such a check would provide a very strong safety net that would virtually eliminate the risk of undetected errors arising from the use of this invention.

Detailed Description Text (79):
We end this section with an example of a narrow hash function that has been implemented and tested and seems to perform well on a variety of data types. The hash function calculates a hash value from three bytes.

Detailed Description Text (98):
Multiple partitionings could also be useful simply to provide a wider pool of subblocks to compare. For example, it may be appropriate to partition each block of data in W different ways using W different functions F.sub.1 . . . F.sub.W where each function yields roughly the same subblock sizes, but at different positions within the block.

Detailed Description Text (100):
Comparing Subblocks

Detailed Description Text (102):
Compare the subblocks themselves.

Detailed Description Text (103):
Compare the hashes of the subblocks.

Detailed Description Text (104):
Compare identifies of the subblocks.

Detailed Description Text (105):
Compare references to the subblocks.

Detailed Description Text (114):
Of the multitude of solutions to the problem of matching blocks of data, one solution is worthy of special attention: the hash table. Hash tables consist of a (usually) finite array of slots into which values may be inserted. To add a value to a hash table, the value is hashed (using a hash function that is usually selected from the class of narrow hash functions) into a slot number, and the value is inserted into that slot. Later, the value can be retrieved in the same manner. Provisions must be made for the case where two data values, to be stored in the same table, hash to the same slot number.

Detailed Description Text (125):
Any component could carry additional checking information such as checksums or digests of the data in the component.

Detailed Description Text (138):
The processing of subblocks defined during a sequential partitioning of a block need not be deferred until the entire block has been partitioned. In particular, the hashes of already-defined subblocks could be calculated and compared while further subblocks are being defined.

Detailed Description Text (151):
Improving the efficiency of the hash calculations by using some part of the calculation of the has of the bytes at one position to calculate the hash at the next position. For example, it may be more efficient to calculate H(x,y,z) if H(*,x,y) has already been calculated. For

example, the Internet IP checksum is organized so that a single running checksum value can be maintained, with bytes entering the window being added to the checksum, and bytes exiting the window being subtracted from the checksum.

Detailed Description Text (157):
This table of hashes can be used to determine if a new subblock is identical to any of the subblocks whose hashes are in the table. To do this, the new subblock's hash is calculated and a check made to see if the hash is in the table.

Detailed Description Text (161):
In one aspect, the invention could be used to determine the broad similarities between two files being compared by a file comparison utility. The utility would partition each of the two files into subblocks, organize the hashes of the subblocks somehow (e.g. using a hash table) to identify all identical subblocks, and then use this information as a framework for reporting similarities and differences between the two files.

Detailed Description Text (163):
If, in addition, a facility was added for recording and comparing the hashes of the entire contents of files and directory trees, a utility could be constructed that could identify all largely similar structures within a file system. Such a utility would be immensely useful when (say) attempting to merge the data on several similar backup tapes.

Detailed Description Text (167):
To perform the backup, E1 compares the hash of Y (stored in S) against the hash of X to see if X has changed (it could also use the modification date file attribute of the file). If X hasn't changed, there is no need to perform any further backup action. If X has changed, E1 partitions X into subblocks, and compares the hashes of these subblocks with the hashes in the shadow file S, so as to find all identical hashes. Identical hashes identify identical subblocks in Y that can be transmitted by reference. E1 then transmits the file as a mixture of raw subblocks and references to subblocks whose hashes appear in S and which are therefore known to appear as subblocks in Y. E1 can also transmit references to subblocks already transmitted. References can take many forms including (without limitation):

Detailed Description Text (176):
The first field contains the MD5 digest (a form of cryptographic hash) of the entire contents of Y. This is included so that it can be copied to the incremental backup file so as to provide a check later that the incremental backup file is not being applied to the wrong version of Y. It could also be used to determine if any change has been made to X since the previous backup Y was taken. The first field is followed by a list of the MD5 digests of the subblocks in Y in the order in which they appear in Y. Finally, a digest of the contents of the shadow file (less this field) is included at the end so as to enable the detection of any corruption of the shadow file.

Detailed Description Text (178):
The first two fields of the incremental backup file contain the MD5 digest of the old and new versions of the file. The hash of the new version X is calculated directly from X. The hash of the old version is obtained from the first field of the shadow file. These two values enable the remote backup entity E2 to check that:

Detailed Description Text (181):
The two checking fields are followed by a list of items followed by a checking digest of the rest of the incremental backup file.

Detailed Description Text (183):
1. The 32-bit index of s subblock in Y. Because E2 possesses Y, it can partition Y itself to construct the same partitioning that was used to create the shadow file. Thus E1 doesn't need to send the hash of any subblock that is in both X and Y. Instead, it need only send the index of the subblock in the list of subblocks constituting Y. This list is represented by the list of hashes in the shadow file S. As 32-bits is wide enough for an index in practice, the saving gained by communicating a 32-bit index instead of a hash is 98 bits for each such item.

Detailed Description Text (191):
Using the first hash in the shadow file to check to see if the entire file has changed at all

before performing the backup process described above.

Detailed Description Text (194):
Although most redundancy in a file system is likely to be found within different versions of each file, there may be great similarities between versions of different files. For example, if a file is renamed, the "new" file will be identical to the "old" file. Such redundancy can e catered for by comparing the hashes of all the files in the old and new versions of a file system. In addition, similarities between different parts of different files can be exploited by comparing the hashes of subblocks of each file to be backed up with the hashes of the subblocks of the entire old version of the file system.

Detailed Description Text (196):
In a further variant, the dependence on the ordering of subblocks could be abandoned, and E1 could simply keep a shadow file containing a list of the hashes of all the subblocks in the previous version (or versions) of the file or file system. E2 would then need to record only a single copy of each unique subblock it has ever received from E1.

Detailed Description Text (201):
The top layer consists of a table of files that binds filenames to lists of subblocks, each list being a list of indexes into the hash table. The reference count of the hash table records the number of references to the subblock that appear in the entire set of files in the file table. The issue of hash table "overflow" can be addressed using a variety of well-known overflow techniques such as that of attaching a linked list to each hash slot.

Detailed Description Text (202):
When a file is read, the list of hash table indexes is converted to pointers to subblocks of data using the hash table. If random access to the file is required, extra information about the length of the subblocks could be added to the file table and/or hash table so as to speed access.

Detailed Description Text (206):
One enhancement that could be made is to exploit unused disk space. Instead of automatically ignoring or overwriting subblocks whose reference count has dropped to zero, the low-redundancy file system could move them to a pool of unused subblocks. These subblocks, while not present in any file, could still form part of the subblock pool referred to when checking to see if incoming subblocks are already present in the file system. The space consumed by subblocks in the unused subblock pool would be recycled only when the disk was full. In the steady state, the "unused" portion of the disk would be filled by subblocks in the unused subblock pool.

Detailed Description Text (216):
The communication application described above considers the case of just two communicants. However, there is no reason why the scheme could not be generalized to cover more than two communicants communicating with each other in private and in public (using broadcasts). For example, to broadcast a block, a computer C.sub.1 could broadcast a list of the hashes of the block's subblocks. Computers C.sub.2 . . . C.sub.N could then each reply indicating which subblocks they do not already possess. C.sub.1 could then broadcast subblocks that many of the other computers do not possess, and send the subblocks missing from only a few computers to those computers privately.

Detailed Description Text (221):
Such a subblock server could be useful for localizing network traffic on the Internet. For example, if a subnetwork (even a large one for (say) an entire country) placed a subblock server on each of its major Internet connections, then (with the appropriate modification of various protocols) much of the traffic into the network could be eliminated. For example, if a user requested a file from a remote host on another network, the user's computer might issue the request and receive, in reply, not the file, but the hashes of the file's subblocks. The user's computer could then send the hashes to the local subblock server to see if the subblocks are present there. It would receive the subblocks that are present and then forward a request for the remaining subblocks to the remote host. The subblock server might notice the new subblocks flowing through it and archive them for future reference. The entire effect could be to eliminate most repeated data transfers between the subnetwork and the rest of the Internet. However, the security implications of schemes such as these would need to be closely investigated before there were deployed.

CLAIMS:

9. The method of claim 1, wherein said subblocks are compared by comparing the hashes of said subblocks.

11. A method for comparing one or more blocks, comprising the steps of:

organizing a block b of digital data for the purpose of comparison, comprising the step of:

partitioning said block b into a plurality of subblocks at at least one position k.vertline.k+1 within said block;

for which b[k-A+1 . . . k+B] satisfies a predetermined constraint; and

wherein A and B are natural numbers,

forming a projection of each said block, being a collection of elements, wherein each element comprises a selected one of a subblock, an identity of a subblock, and a reference of a subblock, and

comparing the elements of said projections of said blocks.

Previous Doc Next Doc Go to Doc#

L21: Entry 3 of 9 File: USPT Nov 23, 2004

DOCUMENT-IDENTIFIER: US 6823519 B1
TITLE: Control object and user interface for controlling networked devices

CLAIMS:

18. In a network comprising a plurality of software controllable devices that communicate over said network, said software controllable devices containing an embedded operating system and a computer-readable storage medium within which information is stored, a system for controlling said software controllable devices, comprising: a plurality of control objects residing in said embedded operating system of respective ones of said software controllable devices, said control objects comprising component object model objects and including logical attributes of said respective ones of said software controllable devices, said control objects further accepting and issuing control messages to and from said respective ones of said software controllable devices, and said control objects being polymorphic such that said control objects are adapted to take on the logical attributes and command and control capabilities of any of said software controllable devices, wherein said control objects register with said system and are assigned an random unique identifier and are active while said respective ones of said software controllable devices are functioning, and wherein said control objects maintain a list of all other registered control objects and their logical attributes and wherein a first registered control object of said plurality of control objects is designated to be a manager object, said manager object performing list management to maintain and administer said list by periodically broadcasting said list to all registered control objects on said system, and wherein said list contains said logical attributes, said random unique identifier and an address of all registered control objects on said system, and wherein any control object in said system can function as said manager object.

25. In a network comprising at least one software controllable device that communicates over said network and is remotely controllable over said network, said software controllable device containing an embedded operating system and a computer-readable storage medium within which state information is stored, a system for controlling said software controllable device, comprising: a control object residing in said embedded operating system and comprising a component object model object, said control object including said state information of said software controllable device, said control object further accepting and issuing control messages to and from said software controllable device, and said control object being polymorphic such that said control object is adapted to take on the logical attributes and command and control capabilities of any software controllable device attached to said network, wherein said control object registers with said system and is active while said software controllable device is functioning, and wherein said control object maintains a list of any other control object registered with said system and corresponding logical attributes, and wherein said control object is designated to be a manager object if it is a first registered object in said network, said manager object performing list management to maintain and administer said list by periodically broadcasting said list to all registered control objects on said network, and wherein said list contains said logical attributes, a random unique identifier and an address of all said registered control objects on said system, and wherein any control object in said system can function as said manager object.

hashing a data file is divided into pieces based on commonality with other pieces in the system using hashing algorithm

CLAIMS:

1. A method for managing data comprising:

producing a probabilistically unique identifier for a digital sequence; and

comparing said probabilistically unique identifier to a list of other identifiers with their corresponding digital sequences.

wherein a prob. unique number is assigned to each of the pieces

2. The method of claim 1 further comprising:

adding said probabilistically unique identifier to said list if said probabilistically unique identifier is not previously in said list.

3. The method of claim 1 further comprising:

removing said probabilistically unique identifier from said list if said probabilistically unique identifier is previously in said list.

and wherein each piece is comb

4. The method of claim 2 further comprising:

adding said digital sequence corresponding to said probabilistically unique identifier to said list.

hash value is

5. The method of claim 3 further comprising:

removing said digital sequence corresponding to said probabilistically unique identifier from said list.

6. The method of claim 4 further comprising:

adding a correspondence between said digital sequence and said probabilistically unique identifier for that sequence.

7. The method of claim 1 wherein said step of producing

hash file

hast function

hash file value

L17: Entry 9 of 9 File: USPT Mar 15, 1994

DOCUMENT-IDENTIFIER: US 5295188 A
TITLE: Public key encryption and decryption circuitry and method

Abstract Text (1):
A high-speed public key cryptosystem is constructed for the encryption and decryption of digital data blocks, the creation and verification of digital signatures, and the creation of verifiable random number sequences. The encryption and decryption techniques employ a public key K constructed as the matrix product of a randomly generated nonsingular matrix M, and a rectangular template matrix T having row identifiers imbedded therein for selecting rows of a matrix M-1 that participate during decryption. Encrypting of a plaintext block, is achieved by converting the block to a nonlinear row selector code used to select rows of K. Columns of the selected rows are then added modulo-2 to produce a block if ciphertext. Deciphering the ciphertext is an iterative process wherein successive row identifiers and plaintext bits are unmasked by selecting rows of matrix M-1 in accordance with the bit states of the ciphertext, and adding columns of the selected rows modulo-2 to produce an intermediate block of data containing 2 plaintext bits and a row identifier used to select a row of key T, which is then added modulo-2 to the intermediate block to reveal the next two bits of plaintext and the next row identifier. A public key consisting of a single, noninvertible, binary matrix and a private key consisting of a pair of binary matrices, one or which is singular and the other nonsingular.

6,219,423

Brief Summary Text (15):
Still further, it is another object of this invention to provide random number sequences that can be verified, if necessary, in the manner of digital signatures.

Detailed Description Text (2):
The subject invention provides a rapid, asymmetric method for achieving reversible and verifiable nonlinear cryptions, or cryptographic transforms, of digital or binary data under control of a pair of transforming tools, one being a series of discrete cryption transforms and the other being a consolidation of the discrete cryption transforms. This consolidation is constructed such that the series of discrete cryption transforms provides a means for achieving the inverse transformation of the consolidated processes and cannot be derived from the consolidated cryption transform. In this sense, the cryption of data described herein is a nonlinear process, meaning that there is no mathematical equation or set of equations that can be used to decrypt data crypted by the instant invention. Such an arrangement of transforming tools, herein implemented as radix 2 or m x n binary matrices, though possible in any radix, can serve many useful data security and verification functions such as public key cryptography, digital signature creation and verification, source message authentication, and random number generation and verification.

Detailed Description Text (10):
Random number sequences useful in a wide variety of applications, including the generation of cryptographic keys themselves and the construction of stream ciphers, may be freely created by submitting a series of inputs (usually a number series) to the decryption process. The resulting output constitutes the random number sequence. For uses such as multi-user arbitration and semaphore establishment in shared network or secure computer environments, such random number sequences or values can be categorically verified in the manner of digital signatures where needed. For example, to establish a precedence between two contending processes A and B of equal priority (where A holds the private key and B holds the corresponding public key), A can submit an input of his choice to B who transforms it with his private key allowing the odd/even parity of the result to resolve the contention. A can verify that B's result was fair and equitable by transforming it by his public key to recover the

input he originally submitted to B.

Detailed Description Text (46):
Additionally, in yet another embodiment, the decryption module may be used as a random number generator, as shown in FIG. 2. Here, any sequence of ordered or nonordered numbers of the requisite bit length stored in serial manner in shift register 300 (dashed lines) may be submitted to buffer memory 44 via lead 302 (dashed lines). A cryptographic transform is then effected as described to obtain a random number set in register 66 of FIG. 3. If desired, this random set of numbers may be verified as being derived from the original sequence of numbers by applying the random set of numbers to the encryption module to obtain the original random numbers.

Detailed Description Text (54):
In the decryption process, row identifiers would be recovered as already noted with the following exception. Each unique row identifier or its corresponding unique complement would identify the row selector code. A typical set of row identifier codes having this property are: 011, 110, and 101. If a row identifier is recovered unchanged, the columns of the A portion of K above it furnish a 000; otherwise, the rows of A furnish a 111, complementing the row identifier. However, if a 000 or a 111 value is obtained, no row identifier is selected. In this manner, the bits of P2 are determined. The bits of P1 are determined in a different manner by first assembling the bits produced by each column of A identified as noted above and inverting these by multiplying them by A.sup.-1 to produce the selecting bits of P1.

CLAIMS:

45. A digital system as set forth in claim 44 further comprising :

block encoding means responsive to a discrete sized sequence of digital data for converting n-length sequential sets of the digital data into a second, nonlinear data stream wherein a particular logic state of each bit each indicates a selection of a discrete row of said second noninvertable matrix;

first matrix storage means for storing said second noninvertable matrix;

first selection means responsive to each said particular logic state of said second data stream, for selecting digits from said second noninvertable matrix, and providing said last-named digits as an output; and

combining means for combining said last-named digits to provide said differently ordered data sequence converted from said ordered sequence of digital data.

46. A digital system as set forth in claim 45 comprising:

third digital generation means for generating an inverse matrix to said invertable digital matrix and providing said inverse matrix as an output;

second matrix storage means for storing said inverse matrix and said second selection means responsive to particular logic states of said differently ordered data sequence, for selecting discrete digital outputs from said second storage means; and

combining means for combining said discrete digital outputs, and providing a second digital sequence as an output.

47. A digital system as set forth in claim 46 wherein said invertable matrix is further characterized by:

a first set of digits on one side of said diagonal and a second set of digits on an opposite side of said diagonal;

third matrix storage means for storing said invertable matrix and sequentially providing selected digits of discrete rows thereof as outputs; and

a plurality of modulo-2 addition means for adding modulo-2, in a sequential manner, selected

L8: Entry 2 of 6 File: PGPB Dec 19, 2002

DOCUMENT-IDENTIFIER: US 20020194484 A1
TITLE: On-disk file format for serverless distributed file system with signed manifest of file modifications

Abstract Paragraph:
In a serverless distributed file system, the writer of a file can provide file authentication information to a verifying machine without having to compute a new digital signature every time a written file is closed. Periodically, the writer compiles a list of the hash values of all files that have been written over a recent interval, computes a hash of the list, and signs the hash. This signed list of hash values is known as a manifest, akin to a shipping manifest that enumerates the items in a shipment. The advantage of using a signed manifest is that the writer need only perform a single signature computation in order to authenticate the writes to multiple files, rather than having to compute a separate signature for each file, as it would if a signature were embedded in each file.

Summary of Invention Paragraph:
[0007] In a serverless distributed file system that stores files across multiple computers, the writer of a file can provide file authentication information to a verifying machine without having to compute a new digital signature every time a written file is closed. Periodically, the writer compiles a list of the hash values of all files that have been written over a recent interval, computes a hash of the list, and signs the hash. This signed list of hash values is known as a "manifest", akin to a shipping manifest that enumerates the items in a shipment. The advantage of using a signed manifest is that the writer need only perform a single signature computation in order to authenticate the writes to multiple files, rather than having to compute a separate signature for each file, as it would if a signature were embedded in each file.

Detail Description Paragraph:
[0037] Generally, according to convergent encryption, a file F (or any other type of encryptable object) is initially hashed using a one-way hashing function h (e.g., SHA, MD5, etc.) to produce a hash value h(F). The file F is then encrypted using a symmetric cipher (e.g., RC4, RC2, etc.) with the hash value as the key, or E.sub.h(F)(F). Next, read access control entries are created for each authorized user who is granted read access to the encrypted file. Write access control is governed by the directory server that stores the directory entry for the file, and it is thus not addressed by the file format and is not discussed further within this document. All references to "access" within this document refer to read access. The access control entries are formed by encrypting the file's hash value h(F) with any number of keys K.sub.1, K.sub.2, . . . , K.sub.m, to yield E.sub.K1(h(F)), E.sub.K2(h(F)), . . . , E.sub.Km(h(F)). In one implementation, each key K is the user's public key of a public/private key pair for an asymmetric cipher (e.g., RSA).

Detail Description Paragraph:
[0039] One advantage of convergent encryption is that the encrypted file can be evaluated by the file system to determine whether it is identical to another file without resorting to any decryption (and hence, without knowledge of any encryption keys). Unwanted duplicative files can be removed by adding the authorized user(s) access control entries to the remaining file. Another advantage is that the access control entries are very small in size, on the order of bytes as compared to possibly gigabytes for the encrypted file. As a result, the amount of overhead information that is stored in each file is small. This enables the property that the total space used to store the file is proportional to the space that is required to store a single encrypted file, plus a constant amount of storage for each additional authorized reader of the file.

Detail Description Paragraph:
[0070] The data stream 402 is designed to allow efficient verification, reading, and writing of portions of the file, without affecting other portions. The data stream is encrypted using the convergent encryption technology described above beneath the heading "File Encryption". For small files, the entire file is hashed and encrypted using the resulting hash value as the encryption key. The encrypted file can be verified without knowledge of the key or any need to decrypt the file first.

Detail Description Paragraph:
[0071] For large files, however, it is difficult to read or update only part of a file because the encryption of the file is based on a hash of the entire file contents. Any write to a file would require re-hashing the entire file followed by re-encrypting with the newly generated hash as the key. Furthermore, verification involves hashing the entire file and examining the hash value. Taking a single hash of the ciphertext of a large file for verification purposes makes writes to part of the file expensive, because any write would once again require hashing the whole file.

Detail Description Paragraph:
[0074] Convergent encryption is then applied to the file at the block level. That is, each block F.sup.i is separately hashed using a one-way hash function (e.g., SHA, MD5, etc.) to produce a hash value h(F.sup.i). Each block F.sup.i is then encrypted using a symmetric cipher (e.g., RC4, RC2, etc.) and the hash value h(F.sup.i) as the key, or E.sub.h(Fi)(F.sup.i). This produces an array 504 of encrypted blocks 506(0)-506(n-1).

Detail Description Paragraph:
[0079] The header 406 contains information pertaining to the file and which may be used to validate the file. In FIG. 4, the header 406 is illustrated as including such file information as a file number 412, a revision number 414, a hash value 416 of the root of tree structure 408, and an optional digital signature 418.

Detail Description Paragraph:
[0081] With the unsigned format, the directory servers send a hash value that represents the contents of the file (as well as its metadata) to a verifying machine for verification of the file contents. In this implementation, there is no way for the verifying machine to determine that a particular user wrote a file, aside from trusting the directory servers. The advantage of this approach is that there is no need to compute or verify digital signatures for the file, which can be computationally expensive. The disadvantage is that the verifying machine must trust the directory servers, although this is less of a problem than it might seem on its face. Even with the signed format, where it is possible to verify that a particular user wrote a file without trusting the directory servers, one still relies on the directory servers to verify that the correct version of the file is present (as opposed to a different file or a old version of the correct file), and to state which user(s) are allowed to sign a particular file. In practice, corrupt directory servers could do much damage even with signed files, so electing to use unsigned files and dropping the signatures saves computational cost at a slight increased risk of reliance on the directory servers.

Detail Description Paragraph:
[0091] The short header form is intended for the extremely common case of very small files that are created once and rarely (or never) overwritten, and readable by either everyone or just the creator. This may be as many as half of all files. The short header form is incompatible with the delegation format, but since these files are created in one piece and then left alone and the delegation format is intended to address in-place updates, the incompatibility is not a problem. The distributed file system is free to decide whether to use the short or long header form for any particular file, and can switch formats on the file (assuming that it has access to the writing user's key).

Detail Description Paragraph:
[0095] The PublicKeyAlgorithm field specifies a suitable public key cipher, such as RSA. The HashAlgorithm field identifies a suitable hash algorithm, such as SHA or MD5. The SymmetricAlgorithm field specifies a suitable symmetric cipher, such as RC2 or RC4, and it employs keys of a size specified in the SymmetricKeySize field (e.g., 128 bit).

Detail Description Paragraph:

[0104] where EncryptedCleartextHash is the access value 604 and the UnencryptedCiphertextHash is the verification value 606. The HASH_SIZE value depends on the HashAlgorithm specified in the header 406. For the SHA algorithm, it is 20 bytes and for the MD5 algorithm, it is 16 bytes. When RC2 encryption is used for the EncryptedCleartextHash, the size is 24 bytes due to padding, regardless of which hash algorithm (MD5 or SHA) is used.

Detail Description Paragraph:

[0105] The existence and size of the tree 408 varies with the size of the file. At one extreme, if the file is less than or equal to one page in size, there is no tree and no per-file secret key K. Instead, the cleartext hash value is turned into a key, encrypted with the public keys of the readers, and stored in the FILE_KEY_ID_PAIR. The hash of the file (that is stored at the directory servers or signed and placed in the file) includes the entire contents of the ciphertext of the file in place of the hash of the highest level that exists in the tree. Since about half of all files are smaller than 4 K, this optimization can be significant.

Detail Description Paragraph:

[0115] The tree root is then hashed to fortn the root node 630, or h(R.sub.X.sup.0). This hash value may then be hashed together with the metadata header 406 and per user information 410 and the resulting hash stored at the directory servers in the case of the unsigned format, or signed using a user's signature in the signed format case. In this way, the hash or signature covers the higher order blocks (of the highest order that exists in the file) and thereby indirectly covers the leaf blocks. The signature covers the used entries in the higher order blocks, not the unused entries and padding. Similarly, the hash entry in the higher order block of the final leaf block does not include any unused entries/padding in that leaf block.

Detail Description Paragraph:

[0123] There are two different types of signatures for a file, depending on how the file is written. In the signed format, the file is signed by the user who is named in the LastWriter field. The signature covers the file header (up to and including the SignatureSize), but does not cover the three offsets so that servers can re-arrange pieces of the Metadata$ stream as they see fit, without having the last writer's key. After the header, the signature then covers the key-id pairs. Following that, it covers either the file ciphertext, the single leaf tree block, or the highest order tree block, depending on the file's size. In the signed format, all that is stored at SignatureOffset is the actual signature blob.

Detail Description Paragraph:

[0129] When a computing device attempts to write a file to the distributed file system 150 and receives a write lock for a file or directory, the computing device generates a random symmetric key, known as the "lock-secret" key. The computing device uses secret sharing to break the lock-secret key into multiple pieces, one piece for each of the directory servers, with a specified number of the servers being sufficient to recover the key.

Detail Description Paragraph:

[0130] If the computing device wants to commit updates to a file without attaching a fill signature to the file (such as on a write-through write to a database file), the computing device generates a delegation certificate and signs the certificate with the user's private key. When the computing device updates a file, it computes the hash of the file that would normally be signed with the writer's private key. However, instead of signing the update, the computing device encrypts it with the lock-secret key using the symmetric signature algorithm specified in the file header.

Detail Description Paragraph:

[0131] If a machine crashes with file updates that are signed with the symmetric signature key (rather than with the normal private-key signature), there will be a set of files signed by lock-secret keys on recovery. For each particular lock-secret key, the computing device takes all files signed by that key and sends the delegation certificates and "symmetric key signatures" to all available directory servers. Once the directory servers have collected all of the appropriate data, they break the seal on the lock-secret key and determine whether the hash of the lock- secret key matches the hash in the DelegationCertificate field. The directory server then decrypt the symmetric key signature (i.e., decrypt the file hash with the lock-secret key) and fill out and sign a DelegationCountersign using the decrypted file hash.

Detail Description Paragraph:

[0135] To check a DelegationCountersign, the verifying computer verifies that SigningMachine is on the list in the DelegationCertificate, that the FileId, FileVersionNumber and DelegationCertificateId match the DelegationCertificate, and that the hash value is the same as the hash value that would have been signed by the last file writer in the normal signed file format.

Detail Description Paragraph:

[0136] There is a related technique for the non-signature case. As before, when a computing device attempts to write a file to the distributed file system 150 and receives a write lock for a file or directory, the computing device generates a symmetric encryption key called the "lock-secret key." The computing device breaks the lock-secret key into multiple pieces and distributes the pieces to the directory servers using a cryptographic secret sharing technique.

Detail Description Paragraph:

[0142] In the first phase, the control module 220 employs the segmenter 222 to divide a file F into "n" multiple blocks 502(0)-502(n-1) at operation 806. Each block contains a portion of the file, which is illustrated as file segments F.sup.0, F.sup.1, F.sup.2, . . . , F.sup.n-1 in blocks 502. At operation 808, the control module 220 invokes the hash module 226 to hash each block 502(0)-502(n-1) to produce intermediate hash values h(F.sup.i). At operation 810, the control module 220 calls the cryptographic engine 224 to encrypt each block 502(0)-502(n-1) using that block's hash value, or E.sub.h(Fi)(F.sup.i). In practice, the hashing and encrypting operations may be accomplished sequentially for each block, one block at a time, before proceeding to the next block. For instance, for each block, a loop may be used to compute the hash of the block, encrypt the result, and then proceed to the next block. With this approach, the two accesses to the block are close together in time, which increases the likelihood that the data for the block will be found in the cache and so be faster to perform. The segmented and encrypted file can then be stored as the unnamed data stream 402.

Detail Description Paragraph:

[0143] During the second phase, the control module 220 uses the tree builder module 230 to construct the block-level access tree. At operation 812, the tree builder 230 (or other module in the client component) generates a random K for the entire file. The tree builder 230 then creates a leaf node L.sup.i for each block 502(0)-502(n-1) (operation 814). Each leaf node L.sup.i contains two components: (1) an access value 604 used for decrypting the corresponding block and (2) a verification value 606 used for verifying the corresponding block. Accordingly, operation 814 can be viewed as two parts. At the first part represented by operation 814(A), the tree builder 230 computes the access value by encrypting the file segment hash h(F.sup.i) using the key K, or E.sub.K(h(F.sup.i)). At the second part represented by operation 814(B), the tree builder 230 computes the verification value by hashing the corresponding encrypted file segment, or h(E.sub.h(Fi)(F.sup.i)).

Detail Description Paragraph:

[0146] Once an effective node structure is created and no more intermediate nodes are desired (i.e., the "no" branch from operation 816), the tree builder 230 forms the root R.sub.X.sup.0 and hashes it to form a hash value h(R.sub.X.sup.0) (operation 822). In the case of the signed format, the control module 220 invokes the signing/verification module 228 to sign the file header 406, per-user information 410 and root node h(R.sub.X.sup.0) with the digital signature of the user identified in the LastWriter field (operation 824). The resultant tree structure 408 is stored in the metadata stream 404. The signature is stored in the header 406 of the metadata stream 404.

Detail Description Paragraph:

[0149] For discussion purposes, suppose that computing device 200 is a verifying machine that is tasked with verifying the first encrypted file block 506(0) for file segment F.sup.0. At operation 902, the signing/verification module 228 evaluates the signature (if any) on the header 406, per-user information 410 and tree root of the tree structure 408 using the public key of the last writer as indicated in the header 406. The signature is held in the header 406 of the metadata stream 404. If the signature is not valid (i.e., the "no" branch from operation 904), the file block is deemed not to be authentic (i.e., block 906). In the non-signed embodiment, the signing/verification module 228 computes the hash that would have been signed in the signed format case, and compares that against the has provided from the directory servers. If the hash does not match, then it follows the "no" branch from operation 904.

Detail Description Paragraph:
[0150] Conversely, if the signature is valid (i.e., the "yes" branch from operation 904), the verification module 228 verifies whether the hash value stored at the root matches the hash of the lower-order nodes below the root in the tree (i.e., operation 908). If the values do not match, the file block is not authentic (i.e., operation 906). If the hash is verified (i.e., the "yes" branch from operation 908), the verification module 228 traverses the tree, node by node, from the root to the leaf node L.sup.0 associated with the target block 506(0). At operation 910, the verification module 228 moves to the next node on the path between the root and the leaf node. If the next node is not a leaf node (i.e., the "no" branch from operation 912), the verification module 228 verifies whether the hash value stored at the next node matches the hash of the lower-order nodes below that node in the tree (i.e., operation 908). In this manner, each node in the path from the root to the leaf node are evaluated. If any one of these verifications fails, the block is not authentic.

Detail Description Paragraph:
[0151] Once the leaf node is reached (i.e., the "yes" branch from block 912), at operation 916, the verification module 228 calls the hash module 226 to compute a hash of the encrypted file segment in target block, or h(E.sub.h(F0)(F.sup.0)). The verification module 228 then compares this resultant hash value with the verification value 606 stored in the corresponding leaf node L.sup.0 (i.e., operation 918). If the two match (i.e., the "yes" branch from operation 920), the target block 506(0) is authentic (operation 922). If the two fail to match (i.e., the "no" branch from operation 920), the target block 506(0) is not authentic (operation 906).

Detail Description Paragraph:
[0156] At operation 1008, the control module 220 calls again on the cryptographic engine 224 to decrypt the target file block using a symmetric cipher D and the recovered hash value as the key, as follows:

Detail Description Paragraph:
[0160] At operation 1102, the computing device modifies a portion of the file contained in block F.sup.i, to create a file block F.sup.i'. Modifying the data renders the previously computed hash value inaccurate and hence unusable. Accordingly, at operation 1104, the control module 220 calls the hash module 226 to compute a new hash value of the modified block, or h (F.sup.i'). At operation 1106, the control module 220 calls the cryptographic engine 224 to encrypt the modified file block F.sup.i, using the new hash value, or E.sub.h(Fi')(F.sup.i'). The new encrypted block replaces the pre-modified encrypted block in the data stream 402.

Detail Description Paragraph:
[0163] In the signed form of the file format, a digital signature is applied to the header 406, per-user information 410 and root node after every modification to the file. This is illustrated, for example, as operation 1112 in the file write process 1100 of FIG. 11. The advantage of the unsigned file format over the signed file format is that the writer of a file does not need to compute a digital signature when closing the file after writing to it. Since digital signatures are computationally expensive, this can be a significant savings if file writes are performed frequently. When using the unsigned file format, instead of signing the file, the writer merely sends the file's hash value to the directory servers that implement the directory in which the file is stored. When another machine wishes to verify the contents of a file, it cannot check a signature in the file, since there is no signature in the file to check. The verifying machine thus needs to obtain verification information from some source that is external to the file.

Detail Description Paragraph:
[0164] One such source is the directory servers that implement the directory in which the file is stored. Since the directory servers store a copy of the file's hash value, they can provide this value to the verifying machine, and the verifying machine can compare this stored hash value to the computed hash value of the file. The disadvantage of this approach is that it requires contacting and trusting the directory servers. The trust issue is not particularly important, since the directory servers already have to be trusted with version information and writer authorizations. However, contacting the directory servers for every file verification can place a significant additional load on these machines, so it is beneficial to avoid this contact if possible.

Detail Description Paragraph:
[0165] Therefore, the present invention includes a mechanism by which the writer of a file can provide file authentication information to a verifying machine without having to compute a new digital signature every time a written file is closed. Periodically, the writer compiles a list of the hash values of all files that have been written over a recent interval, computes a hash of the list, and signs the hash. This signed list of hash values is known as a manifest, analogous to a shipping manifest that enumerates the items in a shipment. The advantage of using a signed manifest is that the writer need only perform a single signature computation in order to authenticate the writes to multiple files, rather than having to compute a separate signature for each file, as it would for the signed file format.

Detail Description Paragraph:
[0166] The writing machine can then send the signed manifest, along with one or more of the files that have been written, to a machine that wants a copy of the files. The receiving machine can verify that the signature of the hash of the manifest is valid, that the hash of manifest is valid, and that the file hash in the manifest corresponds to the hash of the file that it is interested in. The verifying machine needs to know the list of authorized writers to the file, which it must obtain from the directory servers, but this list is generally not modified as frequently as the contents of the file, so the load on the directory servers from propagating updates to the authorized writer list is significantly lower than the load from providing a hash value for every new version of a file.

Detail Description Paragraph:
[0168] At operation 1202, the computing device modifies one or more files. This step is typically performed separately for each file, and depending upon the file size and the scope of the modifications, the control module 220 may invoke one or more of the segmenter 222, the crypto engine 224, the hash module 226, and the tree builder 230 in order to update the file metadata. At operation 1204, the control module 220 calls the hash module 226 to compute a new hash value of each modified file. This step is typically performed separately for each file and in conjunction with the writing of the new data to the file. The control module 220 collects the hash values of every modified file in a manifest (i.e., operation 1206).

Detail Description Paragraph:
[0169] FIG. 13 shows an exemplary manifest 1300. It includes a collection of entries 1302-1306 of modified files. Each entry contains both a file number (i.e., the file number 412 in the file header 406) and the hash of file. The file number specifies to which file a particular hash applies. Also, the manifest 1300 includes a magic number header 1308 at the beginning that helps ascertain what is being signed. This is a different magic number than the one found at the beginning of the file header.

Detail Description Paragraph:
[0171] The manifest may be subsequently verified by initially verifying the signature 1310. If the signature is valid, the file hash contained in the manifest is compared to the hash of the file. If the two match, the verifier will then examine the revision number in the file. Action is only taken if the revision number in the file is bigger than the biggest revision number that the verifier has ever seen for that file. With this last evaluation, the verifier prevents malicious/malfunctioning machines from pushing stale versions of files to replica sites.

Detail Description Paragraph:
[0178] where h(x) is the standard one-way hash function specified in the file format above. The hash function g(x) has the property that data of all zeroes hashes to a hash value of all zeroes.

Detail Description Paragraph:
[0182] Note that the stored hash value for unallocated ciphertext is zero, whereas the stored hash value for zero-value ciphertext is h(0), and the stored hash value for ciphertext corresponding to zero-value plaintext is h(E(0)). Thus, the tree of hashes distinguishes between all-zero primary-stream blocks and unallocated primary-stream blocks. This prevents an attacker from substituting one of these for the other without detection by the directory servers or storage servers. Such a substitution has the ability to affect application behavior, since applications can query the set of allocated ranges in a file.

Detail Description Table CWU:

6 LONGLONG DelegationCertificateOffset; LONGLONG DirectoryServerSignaturesOffset; @ DelegationCertificateOffset is: ULONG Magic; (must be 0.times.dellca7e) UCHAR FormatVersionMajor; (1 described here) UCHAR FormatVersionMinor; (1 described here) USHORT HashedKeySize; Time DelegationTime; GUID FileId; GUID DelegationCertificateId; LONGLONG FileVersionNumber; FILE_USER_NAME LastWriterName; ULONG DirectoryServerCount; ULONG NumDirectoryCOuntersignsNeededForValidity; FILE_MACHINE_NAME DirectoryServer [DirectoryServerCount]; ULONG SignatureSize; <a hash of the secret "signature" symmetric key, of HashedKeySize, using the hash algorithm specified in the file header> <the signature blob of the LastWriter > @ DirectoryServerSignatureOffset is: ULONG CountOfSigningDirectoryServers; for each signing server there is a DelegationCountersign: ULONG Magic (must be 0.times.c2a38452) UCHAR FormatVersionMajor; (1 described here) UCHAR FormatVersionMinor; (1 described here) USHORT HashSize; FARSITE_MACHINE_NAME SigningMachine; GUID FileId; GUID DelegationCertificateId; LONGLONG FileVersionNumber; Time CountersignTime; ULONG SignatureSize; <A hash for the file contents, computed just as the hash that the last writer would sign in the normal signature method, of HashSize> <A signature of the directory server certificate up to but not including SignatureSize, followed by the file contents hash>

CLAIMS:

9. In a distributed file system that stores encrypted files across multiple computers, a method comprising: modifying one or more of the encrypted files; computing a hash value of each modified encrypted file; collecting the hash values into a group; computing a hash value of the group; and digitally signing the hash value of the group of hash values.

10. A method as recited in claim 9, wherein the modified encrypted file includes a metadata stream containing a header and an indexing structure, the indexing structure including hashes of the files and a structure to access the hashes of the files, the computing a hash value of each modified encrypted file further comprising deriving a hash of the header and at least part of the structure.

11. A method as recited in claim 9, wherein the modified encrypted file includes a metadata stream containing a header, per user information, and an indexing tree, the indexing tree including hashes of the files, branch nodes to access the hashes, and a root node, the computing a hash value of each modified encrypted file further comprising hashing as a single composite the header, the per user information, and the root node.

14. One or more computer readable media comprising computer-executable instructions that, when executed, direct a computing device to: modify individual files stored in a serverless distributed file system; compute a hash value of each modified file; collect the hash values into a group; and digitally signing the group of hash values.

15. One or more computer readable media as recited in claim 14, wherein the modified file includes a metadata stream containing a header and an indexing structure, the indexing structure including hashes of the files and a structure to access the hashes of the files, the media further comprising computer-executable instructions that, when executed, direct a computing device to derive a hash of the header and at least part of the structure.

16. One or more computer readable media as recited in claim 14, wherein the modified file includes a metadata stream containing a header, per user information, and an indexing tree, the indexing tree including hashes of the files, branch nodes to access the hashes, and a root node, the media further comprising computer-executable instructions that, when executed, direct a computing device to hash as a single composite the header, the per user information, and the root node.

Previous Doc Next Doc Go to Doc#

L8: Entry 6 of 6 File: USPT Nov 23, 1999

DOCUMENT-IDENTIFIER: US 5990810 A
TITLE: Method for partitioning a block of data into subblocks and for storing and communcating such subblocks

Abstract Text (1):
This invention provides a method and apparatus for detecting common spans within one or more data blocks by partitioning the blocks (FIG. 4) into subblocks and searching the group of subblocks (FIG. 12) (or their corresponding hashes (FIG. 13)) for duplicates. Blocks can be partitioned into subblocks using a variety of methods, including methods that place subblock boundaries at fixed positions (FIG. 3), methods that place subblock boundaries at data-dependent positions (FIG. 3), and methods that yield multiple overlapping subblocks (FIG. 6). By comparing the hashes of subblocks, common spans of one or more blocks can be identified without ever having to compare the blocks or subblocks themselves (FIG. 13). This leads to several applications including an incremental backup system that backs up changes rather than changed files (FIG. 25), a utility that determines the similarities and differences between two files (FIG. 13), a file system that stores each unique subblock at most once (FIG. 26), and a communications system that eliminates the need to transmit subblocks already possessed by the receiver (FIG. 19).

Brief Summary Text (8):
To identify identical portions of data within a group of blocks of data, the blocks must be analysed. One simple approach is to divide the blocks into fixed-length (e.g. 512-byte) subblocks and compare these with each other so as to identify all identical subblocks. This knowledge can the be used to manage the blocks in more efficient ways.

Brief Summary Text (19):
Low-redundancy file system: Data stored in a file system can be partitioned into subblocks whose hashes can be compared so as to eliminate the redundant storage of identical subblocks.

Brief Summary Text (22):
The term block and subblock both refer, without limitation, to finite blocks or infinite blocks (sometimes called streams) of zero or more bits or bytes of digital data. Although the two different terms ("blocks" and "subblock") essentially describe the same substance (digital data), the two different terms have been employed in this specification to indicate the role that a particular piece of data is playing. The term "block" is usually used to refer to raw data to be manipulated by aspects of the invention. The term "subblock" is usually used to refer to a part of a block. "Blocks" are "partitioned" into "subblocks".

Brief Summary Text (44):
Note: In most applications, the output of this aspect will be an ordered list of hashes of the subblocks of the block.

Brief Summary Text (45):
9. In a further aspect of the invention, the invention provides a method for comparing one or more blocks, comprising the steps of:

Brief Summary Text (48):
c. Comparing the elements of said projections of said blocks.

Brief Summary Text (64):
Note: It is implicit in this aspect the E1 will be able to use comparison (or other methods) to use its knowledge of E2's possession of Y to determine the set of subblocks that are common to

this class: a 64-bit DES hash.

Detailed Description Text (50):
Strong one-way has functions: Strong one-way hash functions are the same as weak one-way hash functions except that they have the additional property of providing cryptographic assurance that it is difficult to find any two different blocks that have the same hash value, where the hash value is unspecified. Examples of this class: MD4, MD5, and SHA-1.

Detailed Description Text (56):
Depending on the application, hash functions from any of the four classes above could be employed in either role. However, as the determination of subblock boundaries does not require identity or cryptographic strength, it would be inefficient to use hash functions from any but the weakest class. Similarly, the need for identity, the ever-present threat of subversion, and the minor performance penalty for strong one-way hash functions (compared to weak ones) suggests that nothing less than strong one-way hash functions should be used to calculate subblock identities.

Detailed Description Text (61):
The theoretical properties of cryptographic hashes (and here is meant strong one-way hash functions) yield particularly interesting practical properties. Because such hashes are significantly wide, the probability of two randomly-chosen subblocks having the same hash is practically zero (for a 128-bit hash, it is about one in 10.sup.38), and because it is computationally infeasible to find two subblocks having the same hash, it is practically guaranteed that no intelligent agent will be able to do so. The implication of these properties is that from a practical perspective, the finite set of hash values for a particular cryptographic hash algorithm is one-to-one with the infinite set of finite variable length subblocks. This theoretically impossible property manifests itself in practice because of the practical infeasibility of finding two subblocks that hash to the same value.

Detailed Description Text (62):
This property means that, for the purposes of comparison (for identically), cryptographic hashes may safely be used in place of the subblocks from which they were calculated. As most cryptographic hashes are only about 128 bits long, hashes provide an extremely efficient way to compare subblocks without requiring the direct comparison of the content of the subblocks themselves. Such comparisons can be used to eliminate many transmissions of information. For example, a subblock X.sub.1 on a computer C1 in Sydney could be compared with a subblock Y.sub.1 on a computer C2 in Boston by a computer C3 in Paris, with the total theoretical network traffic being just 256 bits (C1 and C2 each send the 128-bit hash of their respective subblocks to C3 for comparison, and C3 compares the two hashes).

Detailed Description Text (64):
Cryptographic hashes can be used to compare two subblocks without having to compare, or requiring access to, the content of the subblocks.

Detailed Description Text (65):
If it is necessary to be able to determine whether a subblock T is identical to one of a group of subblocks, the subblocks themselves need not be stored, just a collection of their hashes. The hash of any candidate subblock can then be compared with the hashes in the collection to establish whether the subblock is in the group of subblocks from which the collection of hashes was generated.

Detailed Description Text (66):
Cryptographic hashes can be used to ensure that the partitioning of a block into subblocks and the subsequent reassembly of the subblocks into a reconstructed block is error-free. This can be done by comparing the hash of the original block with the hash of the reconstructed block.

Detailed Description Text (68):
If an entity E1 passes a key (consisting of a block of bits) chosen at random to an entity E2, E2 may then prove to E1 that it possesses a subblock by sending E1 the hash of the concatenation of the key and the subblock. This mechanism could be used as an additional check in security applications.

Detailed Description Text (69):

example, the Internet IP checksum is organized so that a single running checksum value can be maintained, with bytes entering the window being added to the checksum, and bytes exiting the window being subtracted from the checksum.

Detailed Description Text (157):
This table of hashes can be used to determine if a new subblock is identical to any of the subblocks whose hashes are in the table. To do this, the new subblock's hash is calculated and a check made to see if the hash is in the table.

Detailed Description Text (161):
In one aspect, the invention could be used to determine the broad similarities between two files being compared by a file comparison utility. The utility would partition each of the two files into subblocks, organize the hashes of the subblocks somehow (e.g. using a hash table) to identify all identical subblocks, and then use this information as a framework for reporting similarities and differences between the two files.

Detailed Description Text (163):
If, in addition, a facility was added for recording and comparing the hashes of the entire contents of files and directory trees, a utility could be constructed that could identify all largely similar structures within a file system. Such a utility would be immensely useful when (say) attempting to merge the data on several similar backup tapes.

Detailed Description Text (167):
To perform the backup, E1 compares the hash of Y (stored in S) against the hash of X to see if X has changed (it could also use the modification date file attribute of the file). If X hasn't changed, there is no need to perform any further backup action. If X has changed, E1 partitions X into subblocks, and compares the hashes of these subblocks with the hashes in the shadow file S, so as to find all identical hashes. Identical hashes identify identical subblocks in Y that can be transmitted by reference. E1 then transmits the file as a mixture of raw subblocks and references to subblocks whose hashes appear in S and which are therefore known to appear as subblocks in Y. E1 can also transmit references to subblocks already transmitted. References can take many forms including (without limitation):

Detailed Description Text (176):
The first field contains the MD5 digest (a form of cryptographic hash) of the entire contents of Y. This is included so that it can be copied to the incremental backup file so as to provide a check later that the incremental backup file is not being applied to the wrong version of Y. It could also be used to determine if any change has been made to X since the previous backup Y was taken. The first field is followed by a list of the MD5 digests of the subblocks in Y in the order in which they appear in Y. Finally, a digest of the contents of the shadow file (less this field) is included at the end so as to enable the detection of any corruption of the shadow file.

Detailed Description Text (178):
The first two fields of the incremental backup file contain the MD5 digest of the old and new versions of the file. The hash of the new version X is calculated directly from X. The hash of the old version is obtained from the first field of the shadow file. These two values enable the remote backup entity E2 to check that:

Detailed Description Text (181):
The two checking fields are followed by a list of items followed by a checking digest of the rest of the incremental backup file.

Detailed Description Text (183):
1. The 32-bit index of s subblock in Y. Because E2 possesses Y, it can partition Y itself to construct the same partitioning that was used to create the shadow file. Thus E1 doesn't need to send the hash of any subblock that is in both X and Y. Instead, it need only send the index of the subblock in the list of subblocks constituting Y. This list is represented by the list of hashes in the shadow file S. As 32-bits is wide enough for an index in practice, the saving gained by communicating a 32-bit index instead of a hash is 98 bits for each such item.

Detailed Description Text (191):
Using the first hash in the shadow file to check to see if the entire file has changed at all

before performing the backup process described above.

Detailed Description Text (194):
Although most redundancy in a file system is likely to be found within different versions of each file, there may be great similarities between versions of different files. For example, if a file is renamed, the "new" file will be identical to the "old" file. Such redundancy can e catered for by comparing the hashes of all the files in the old and new versions of a file system. In addition, similarities between different parts of different files can be exploited by comparing the hashes of subblocks of each file to be backed up with the hashes of the subblocks of the entire old version of the file system.

Detailed Description Text (196):
In a further variant, the dependence on the ordering of subblocks could be abandoned, and E1 could simply keep a shadow file containing a list of the hashes of all the subblocks in the previous version (or versions) of the file or file system. E2 would then need to record only a single copy of each unique subblock it has ever received from E1.

Detailed Description Text (201):
The top layer consists of a table of files that binds filenames to lists of subblocks, each list being a list of indexes into the hash table. The reference count of the hash table records the number of references to the subblock that appear in the entire set of files in the file table. The issue of hash table "overflow" can be addressed using a variety of well-known overflow techniques such as that of attaching a linked list to each hash slot.

Detailed Description Text (202):
When a file is read, the list of hash table indexes is converted to pointers to subblocks of data using the hash table. If random access to the file is required, extra information about the length of the subblocks could be added to the file table and/or hash table so as to speed access.

Detailed Description Text (206):
One enhancement that could be made is to exploit unused disk space. Instead of automatically ignoring or overwriting subblocks whose reference count has dropped to zero, the low-redundancy file system could move them to a pool of unused subblocks. These subblocks, while not present in any file, could still form part of the subblock pool referred to when checking to see if incoming subblocks are already present in the file system. The space consumed by subblocks in the unused subblock pool would be recycled only when the disk was full. In the steady state, the "unused" portion of the disk would be filled by subblocks in the unused subblock pool.

Detailed Description Text (216):
The communication application described above considers the case of just two communicants. However, there is no reason why the scheme could not be generalized to cover more than two communicants communicating with each other in private and in public (using broadcasts). For example, to broadcast a block, a computer C.sub.1 could broadcast a list of the hashes of the block's subblocks. Computers C.sub.2 . . . C.sub.N could then each reply indicating which subblocks they do not already possess. C.sub.1 could then broadcast subblocks that many of the other computers do not possess, and send the subblocks missing from only a few computers to those computers privately.

Detailed Description Text (221):
Such a subblock server could be useful for localizing network traffic on the Internet. For example, if a subnetwork (even a large one for (say) an entire country) placed a subblock server on each of its major Internet connections, then (with the appropriate modification of various protocols) much of the traffic into the network could be eliminated. For example, if a user requested a file from a remote host on another network, the user's computer might issue the request and receive, in reply, not the file, but the hashes of the file's subblocks. The user's computer could then send the hashes to the local subblock server to see if the subblocks are present there. It would receive the subblocks that are present and then forward a request for the remaining subblocks to the remote host. The subblock server might notice the new subblocks flowing through it and archive them for future reference. The entire effect could be to eliminate most repeated data transfers between the subnetwork and the rest of the Internet. However, the security implications of schemes such as these would need to be closely investigated before there were deployed.

CLAIMS:

9. The method of claim 1, wherein said subblocks are compared by comparing the hashes of said subblocks.

11. A method for comparing one or more blocks, comprising the steps of:

organizing a block b of digital data for the purpose of comparison, comprising the step of:

partitioning said block b into a plurality of subblocks at at least one position k.vertline.k+1 within said block;

for which b[k-A+1 . . . k+B] satisfies a predetermined constraint; and

wherein A and B are natural numbers,

forming a projection of each said block, being a collection of elements, wherein each element comprises a selected one of a subblock, an identity of a subblock, and a reference of a subblock, and

comparing the elements of said projections of said blocks.

Previous Doc Next Doc Go to Doc#

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 6 of 6 returned.

☐ 1. Document ID: US 20040148306 A1

Using default format because multiple data bases are involved.

L8: Entry 1 of 6 File: PGPB Jul 29, 2004

PGPUB-DOCUMENT-NUMBER: 20040148306
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20040148306 A1

TITLE: Hash file system and method for use in a commonality factoring system

PUBLICATION-DATE: July 29, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Moulton, Gregory Hagan	Irvine	CA	US	
Whitehill, Stephen B.	Tustin	CA	US	

US-CL-CURRENT: 707/101; 711/114

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw. Desc | Image

☐ 2. Document ID: US 20020194484 A1

L8: Entry 2 of 6 File: PGPB Dec 19, 2002

PGPUB-DOCUMENT-NUMBER: 20020194484
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020194484 A1

TITLE: On-disk file format for serverless distributed file system with signed manifest of file modifications

PUBLICATION-DATE: December 19, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Bolosky, William J.	Issaquah	WA	US	
Adya, Atul	Bellevue	WA	US	
Douceur, John R.	Bellevue	WA	US	

US-CL-CURRENT: 713/189

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw. Desc | Image